



Task Based Programming on Embedded Multicores

Schleuniger, Pascal; Karlsson, Sven

Publication date:
2013

Document Version
Publisher's PDF, also known as Version of record

[Link back to DTU Orbit](#)

Citation (APA):
Schleuniger, P., & Karlsson, S. (2013). *Task Based Programming on Embedded Multicores*. Poster session presented at 8th International Conference on High-Performance and Embedded Architectures and Compilers , Berlin, Germany.

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Task Based Programming on Embedded Multicores

Pascal Schleuniger and Sven Karlsson
Technical University of Denmark



Motivation

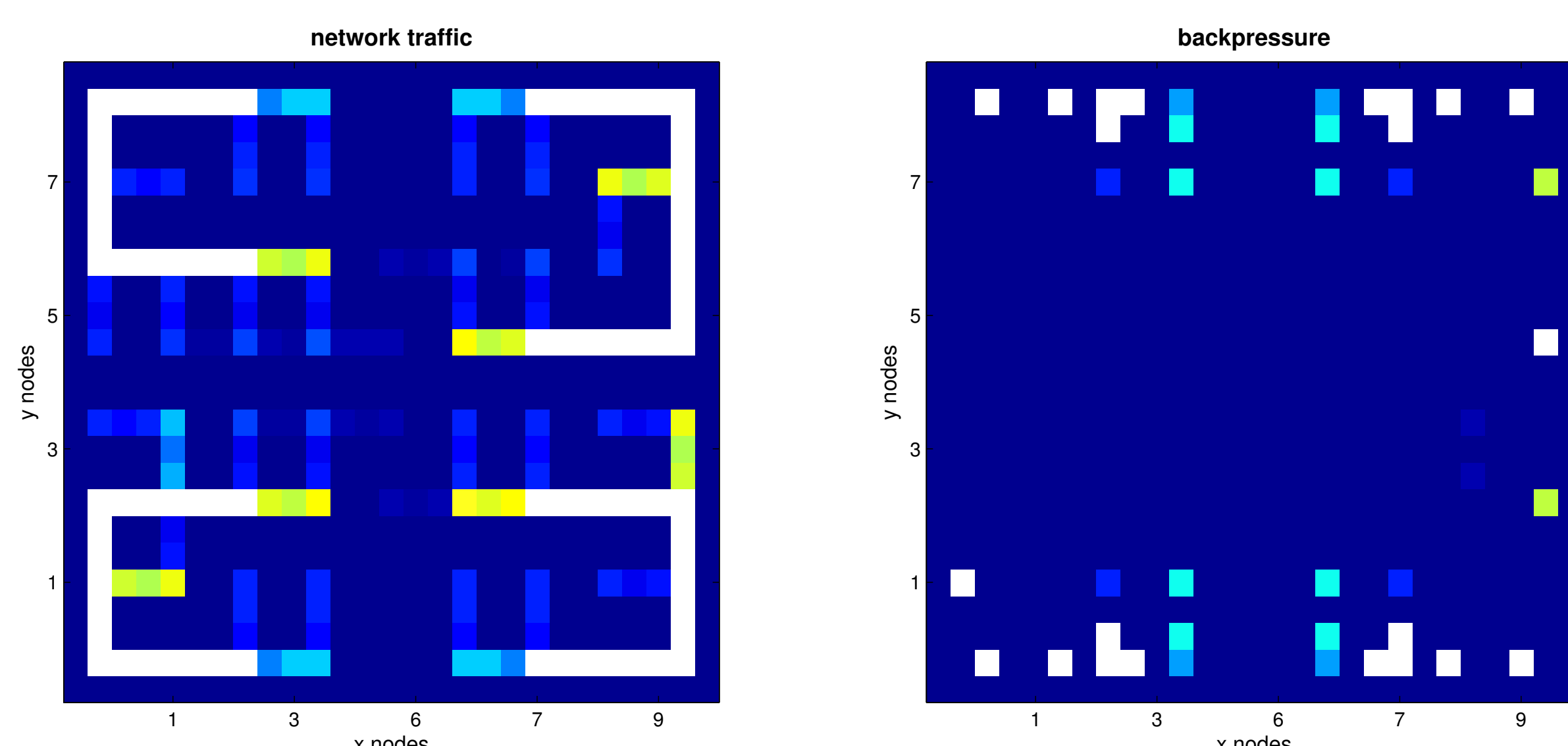
- ▶ Directory based cache coherency protocols have drawbacks:
 - ▶ They introduce a high communication overhead.
 - ▶ Induced latency limits the scalability of the system.
 - ▶ Directories may occupy up to 20% of the total memory.
 - ▶ Low power efficiency.
 - ▶ High design and implementation complexity. State machines have a multitude of transitional states.
- ▶ We argue that parallelism should be expressed using a task based model. We also claim this will simplify the cache coherency protocol.

Contributions

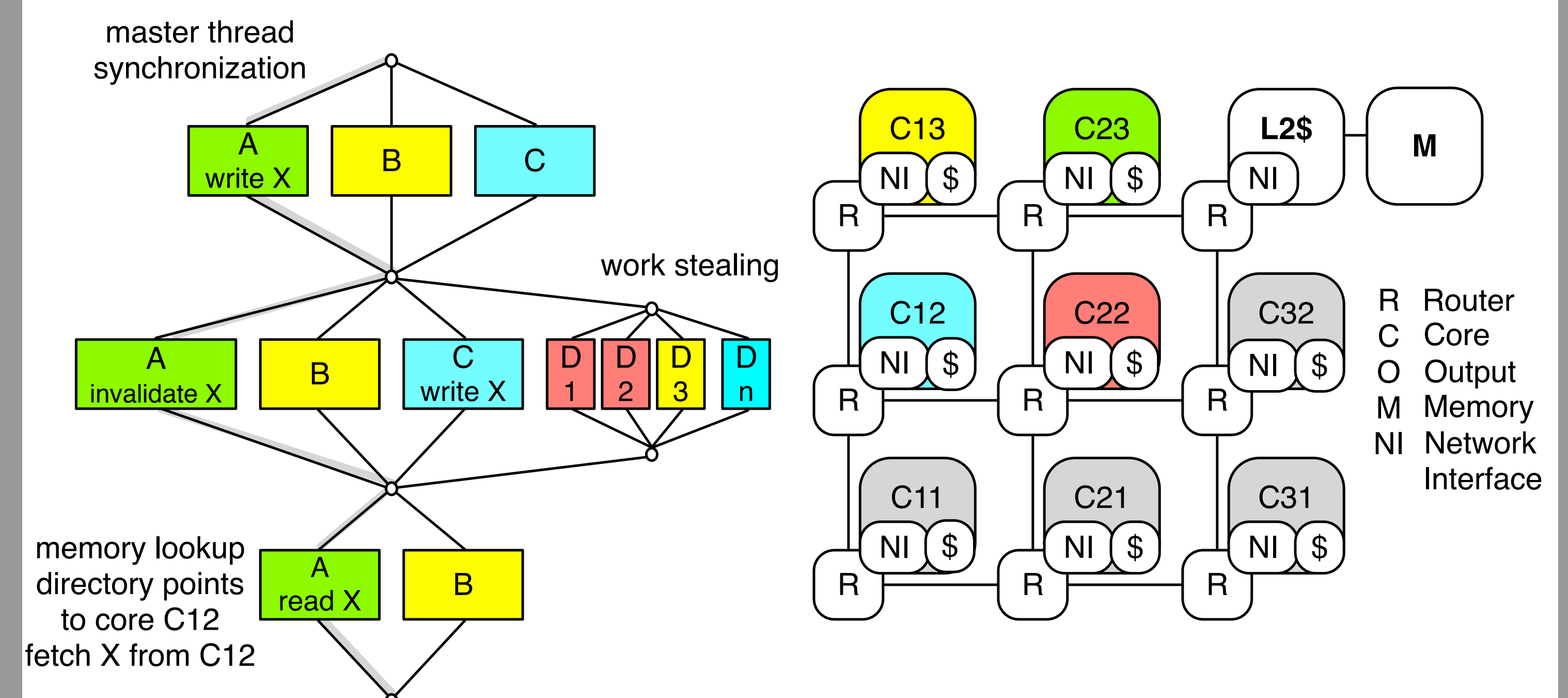
- ▶ We design a scalable high performance multicore platform on FPGA.
- ▶ We outline the runtime environment needed to support task models.
- ▶ We only do cache coherency operations at task boundaries to simplify the cache coherency protocol.

Architecture

- ▶ Processor Core
 - ▶ Tinsu processor core optimized for a high throughput on FPGA.
 - ▶ 8-stage single issue, in-order pipeline, support of predicated instructions.
 - ▶ Support both hard- and soft float operations.
 - ▶ Full GCC based tool suite: GCC, Binutils tools, and NewlibC library
- ▶ Network Interconnect
 - ▶ Generic 2D mesh on-chip network optimized for FPGA implementation.
 - ▶ Packet switched, deadlock free XY-routing scheme.
 - ▶ 1 cycle latency per network hop.
 - ▶ Peak switching data rate of 9.6 Gbits/s per link.
- ▶ Simulation Environment
 - ▶ Platform independent behavior level VHDL.
 - ▶ Full system simulator with the GHDL open source VHDL compiler.
 - ▶ Simulator runs ELF executables.
 - ▶ Simulation speed: 1 kHz for single core / 10 Hz for a 64 core system.
 - ▶ Allows for monitoring and plot network traffic and CPU utilization.



Task Model Example



Task semantics:

- ▶ C extensions to spawn and synchronize tasks.
- ▶ Use "spawn" keyword to create a number of parallel tasks.
- ▶ Each core has its own queue of tasks.
- ▶ Spawned tasks are put on the top of the spawning core's task queue.
- ▶ Idle cores steal work from the bottom of the task queues of other cores.
- ▶ Use "sync" keyword to wait for parallel tasks to finish.
- ▶ Nested tasks are possible.

Memory Consistency Model:

- ▶ Only task stealing leads to coherency actions.
- ▶ Between parallel tasks, memory is not kept coherent.
- ▶ In a set of parallel tasks, there can only be either a single reader and writer of a memory location or multiple readers. It is up to the programmer to assure this.
- ▶ If and only if a stolen task finishes, memory coherency actions take place.
- ▶ Sync operations only complete once memory coherency actions have finished.
- ▶ Hardware support for synchronization primitives to avoid spin-locks.

Implementation:

- ▶ Support of "load-linked" and "store conditional" operations to steal tasks and to synchronize.

Conclusions

- ▶ We design and implement a scalable high performance multicore platform on FPGA.
- ▶ We outline the runtime environment for tasks:
 - ▶ global shared address space
 - ▶ cache coherency operations are only done at task boundaries to simplify the cache coherency protocol
- ▶ We envision hardware support for synchronization primitives to avoid spin-locks.

References

- ▶ M. Frigo et al. The implementation of the Cilk-5 multithreaded language. In PLDI '98, pages 212-223, 1998
- ▶ B. Choi et al. DeNovo: Rethinking the Memory Hierarchy for Disciplined Parallelism. In PACT '11, pages 155-166, 2011